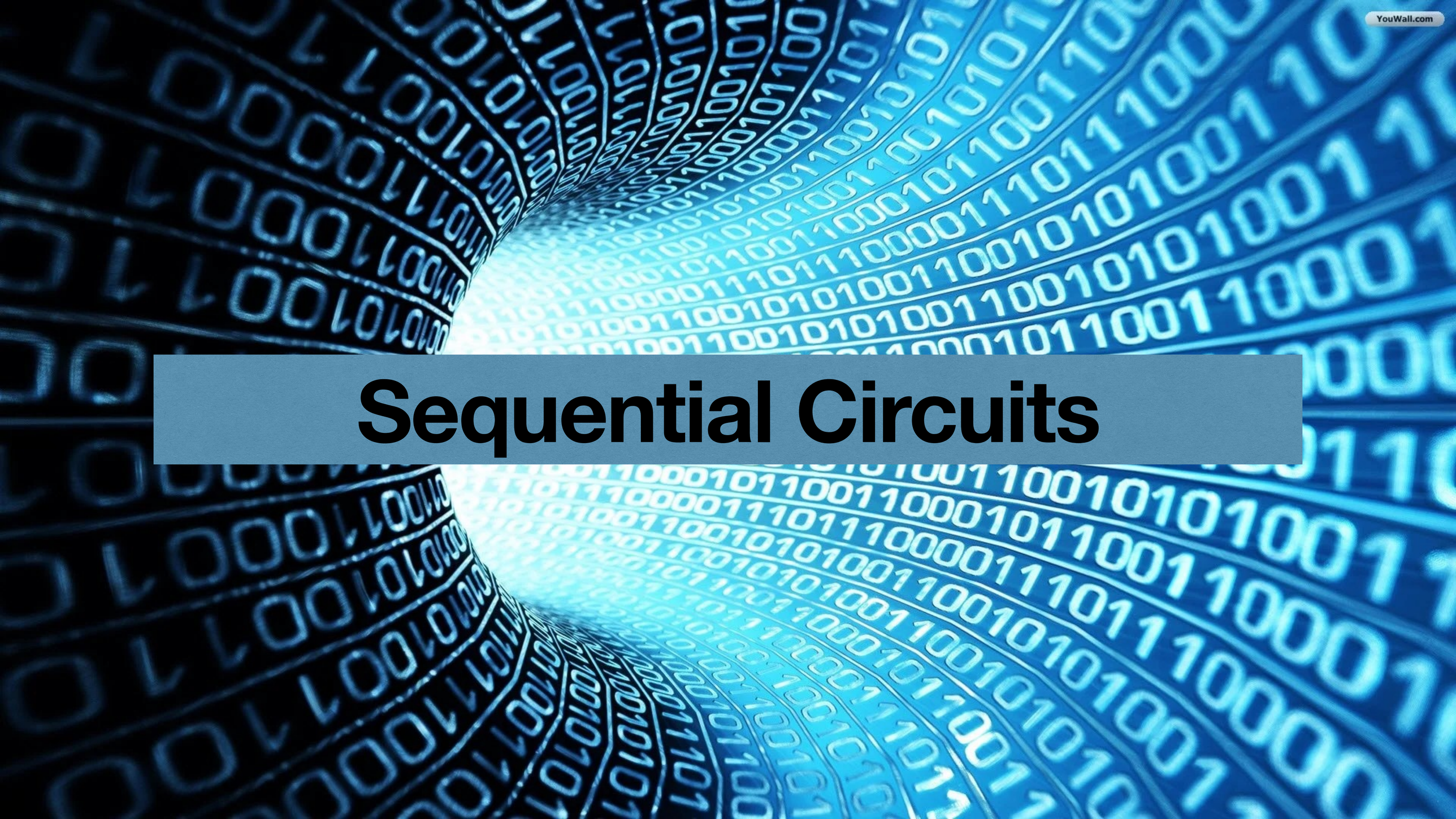
Digital Logic Design + Computer Architecture

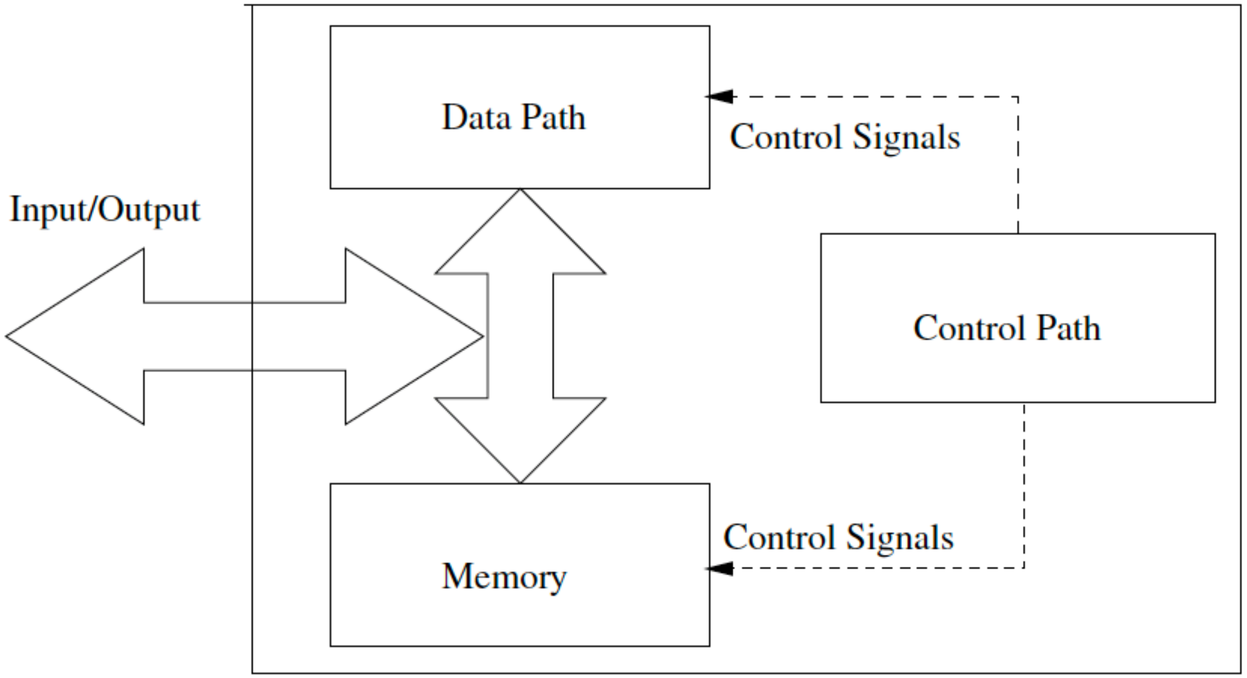
Sayandeep Saha

Assistant Professor
Department of Computer Science and Engineering
Indian Institute of Technology Bombay



Sequential Circuits

General View of a Hardware: Once Again



Binary GCD Algorithm

Input: Integers u and v
Output: Greatest Common Divisor of u and v: $z = gcd(u, v)$

```
1  while (u!=v) do
2      if u and v are even then
3          z = 2gcd(u/2, v/2)
4      end
5      else if (u is odd and v is even) then
6          z = gcd(u, v/2)
7      end
8      else if (u is even and v is odd) then
9          z = gcd(u/2, v)
10     end
11     else
12         if (u ≥ v) then
13             z = gcd((u − v)/2, v)
14         end
15         else
16             z = gcd(u, (v − u)/2)
17         end
18     end
19 end
```

Binary GCD Algorithm: Closer to Hardware

Input: Integers u and v
Output: Greatest Common Divisor of u and v: $z = gcd(u, v)$

```
1  while (u!=v) do
2      if u and v are even then
3          z = 2gcd(u/2, v/2)
4      end
5      else if (u is odd and v is even) then
6          z = gcd(u, v/2)
7      end
8      else if (u is even and v is odd) then
9          z = gcd(u/2, v)
10     end
11     else
12         if (u ≥ v) then
13             z = gcd((u − v)/2, v)
14         end
15         else
16             z = gcd(u, (v − u)/2)
17         end
18     end
19 end
```

Input: Integers u and v
Output: Greatest Common Divisor of u and v: $z = gcd(u, v)$

```
1  register X_R, Y_R;
2  X_R = u; Y_R = v; count = 0;
3  while (X_R! = Y_R) do
4      if (X_R[0] = 0 and Y_R[0] = 0) then
5          X_R = right shift(X_R)
6          Y_R = right shift(Y_R)
7          count = count + 1
8      end
9      else if (X_R[0] = 1 and Y_R[0] = 1) then
10         Y_R = right shift(Y_R)
11     end
12     else if (X_R[0] = 0 and Y_R[0] = 1) then
13         X_R = right shift(X_R)
14     end
15     else
16         if (X_R ≥ Y_R) then
17             X_R = right shift(X_R − Y_R)
18         end
19         else
20             Y_R = right shift(Y_R − X_R)
21         end
22 end
23 while (count > 0) do
24     X_R = left shift(X_R)
25     count = count − 1
26 end
```

Binary GCD Algorithm: Closer to Hardware

Input: Integers u and v
Output: Greatest Common Divisor of u and v: $z = gcd(u, v)$

```
1  while (u!=v) do
2      if u and v are even then
3          z = 2gcd(u/2, v/2)
4      end
5      else if (u is odd and v is even) then
6          z = gcd(u, v/2)
7      end
8      else if (u is even and v is odd) then
9          z = gcd(u/2, v)
10     end
11     else
12         if (u ≥ v) then
13             z = gcd((u − v)/2, v)
14         end
15         else
16             z = gcd(u, (v − u)/2)
17         end
18     end
19 end
```

How to interpret as hardware?

Keep the count of the 2's getting multiplied

Input: Integers u and v
Output: Greatest Common Divisor of u and v: $z = gcd(u, v)$

```
1  register X_R, Y_R;
2  X_R = u; Y_R = v; count = 0;
3  while (X_R! = Y_R) do
4      if (X_R[0] = 0 and Y_R[0] = 0) then
5          X_R = right shift(X_R)
6          Y_R = right shift(Y_R)
7          count = count + 1
8      end
9      else if (X_R[0] = 1 and Y_R[0] = 1) then
10         Y_R = right shift(Y_R)
11     end
12     else if (X_R[0] = 0 and Y_R[0] = 1) then
13         X_R = right shift(X_R)
14     end
15     else
16         if (X_R ≥ Y_R) then
17             X_R = right shift(X_R − Y_R)
18         end
19         else
20             Y_R = right shift(Y_R − X_R)
21         end
22 end
23 while (count > 0) do
24     X_R = left shift(X_R)
25     count = count − 1
26 end
```

Constructing the Datapath

Input: Integers u and v
Output: Greatest Common Divisor of u and v: $z = gcd(u, v)$

```
1  register X_R, Y_R;
2  X_R = u; Y_R = v; count = 0;
3  while (X_R != Y_R) do
4      if (X_R[0] = 0 and Y_R[0] = 0) then
5          X_R = right shift(X_R)
6          Y_R = right shift(Y_R)
7          count = count + 1
8      end
9      else if (X_R[0] = 1 and Y_R[0] = 1) then
10         Y_R = right shift(Y_R)
11     end
12     else if (X_R[0] = 0 and Y_R[0] = 1) then
13         X_R = right shift(X_R)
14     end
15     else
16         if (X_R ≥ Y_R) then
17             X_R = right shift(X_R − Y_R)
18         end
19         else
20             Y_R = right shift(Y_R − X_R)
21         end
22 end
23 while (count > 0) do
24     X_R = left shift(X_R)
25     count = count − 1
26 end
```

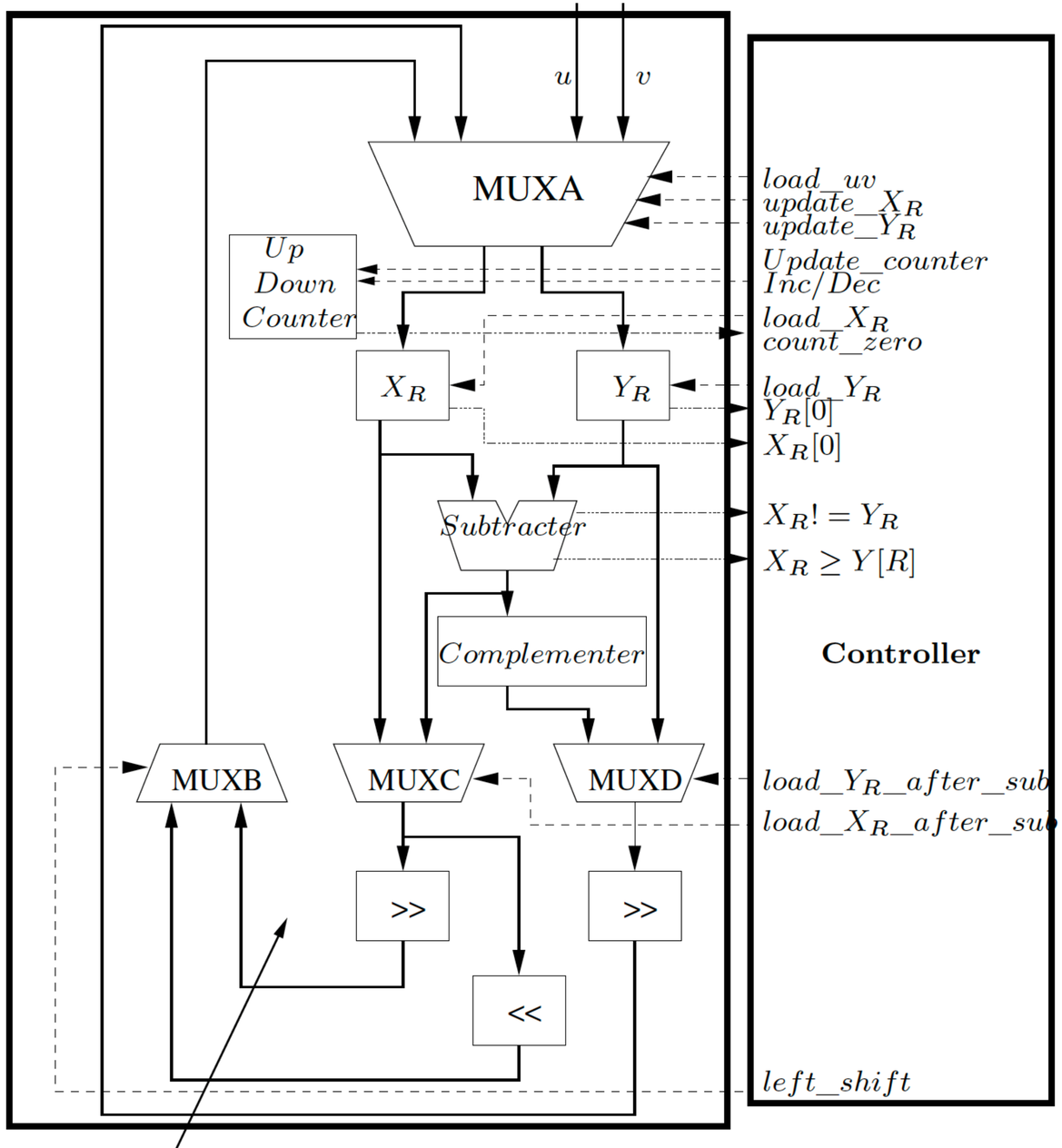
- **Required hardware components**
  - Two Registers for holding u and v (sequential)
- **Datapath elements**
  - Subtractor
  - Complementer
  - Right shifter
  - Left shifter
  - Counter (sequential)
  - Multiplexors — to route the control signals

Constructing the Controller

```
1  register X_R, Y_R;
2  X_R = u; Y_R = v; count = 0;                          /* State 0 */
3  while (X_R != Y_R) do
4      if (X_R[0] = 0 and Y_R[0] = 0) then               /* State 1 */
5          X_R = right shift(X_R)
6          Y_R = right shift(Y_R)
7          count = count + 1
8      end
9      else if (X_R[0] = 1 and Y_R[0] = 1) then          /* State 2 */
10         Y_R = right shift(Y_R)
11     end
12     else if (X_R[0] = 0 and Y_R[0] = 1) then          /* State 3 */
13         X_R = right shift(X_R)
14     end
15     else                                              /* State 4 */
16         if (X_R ≥ Y_R) then
17             X_R = right shift(X_R − Y_R)
18         end
19         else
20             Y_R = right shift(Y_R − X_R)
21         end
22 end
23 while (count > 0) do                                  /* State 5 */
24     X_R = left shift(X_R)
25     count = count − 1
26 end
```

- **Observation**
  - Whenever there is an if-else block, we allocate a state
  - Why? Because they dictate the control flow
  - The while loop is handled by the counter
    - Give the counter value to the controller and it tells you when to stop
  - **Note**: there might be cases when you have no if-else statements, but you have to sequentialize the computation
    - Remember the 4a+b example.
    - Whenever you need to sequentialise, allocate states.

Constructing the Hardware Diagram



Input: Integers u and v
Output: Greatest Common Divisor of u and v: $z = gcd(u, v)$

```
1  register X_R, Y_R;
2  X_R = u; Y_R = v; count = 0;
3  while (X_R != Y_R) do
4      if (X_R[0] = 0 and Y_R[0] = 0) then
5          X_R = right shift(X_R)
6          Y_R = right shift(Y_R)
7          count = count + 1
8      end
9      else if (X_R[0] = 1 and Y_R[0] = 1) then
10         Y_R = right shift(Y_R)
11     end
12     else if (X_R[0] = 0 and Y_R[0] = 1) then
13         X_R = right shift(X_R)
14     end
15     else
16         if (X_R ≥ Y_R) then
17             X_R = right shift(X_R − Y_R)
18         end
19         else
20             Y_R = right shift(Y_R − X_R)
21         end
22 end
23 while (count > 0) do
24     X_R = left shift(X_R)
25     count = count − 1
26 end
```

Constructing the State Transition Table

Present State	Next State					Output Signals										
	0____	100_	110_	101_	111_	*load uv*	*update* X_R	*update* Y_R	*load* X_R	*load* Y_R	*load*_X_R *after*_*sub*	*load*_Y_R *after*_*sub*	*Update counter*	*Inc /Dec*	*left shift*	*count zero*
S_0	S_5	S_1	S_2	S_3	S_4	1	0	0	1	1	0	0	0	_	_	_
S_1	S_5	S_1	S_2	S_3	S_4	0	1	1	1	1	0	0	1	1	_	_
S_2	S_5	S_1	S_2	S_3	S_4	0	0	1	0	1	0	0	0	_	_	_
S_3	S_5	S_1	S_2	S_3	S_4	0	1	0	1	0	0	0	0	_	_	_
S_4 $(X_R \geq Y_R)$	S_5	S_1	S_2	S_3	S_4	0	1	0	1	0	1	0	0	_	_	_
S_4 $(X_R < Y_R)$	S_5	S_1	S_2	S_3	S_4	0	0	1	0	1	0	1	0	_	_	_
S_5	S_5	S_5	S_5	S_5	S_5	0	0	0	0	0	0	0	1	0	1	0

- **Controller receives 4 input signals**
  - XR != YR
  - XR[0]
  - YR[0]
  - XR >= YR

Now go and code it down